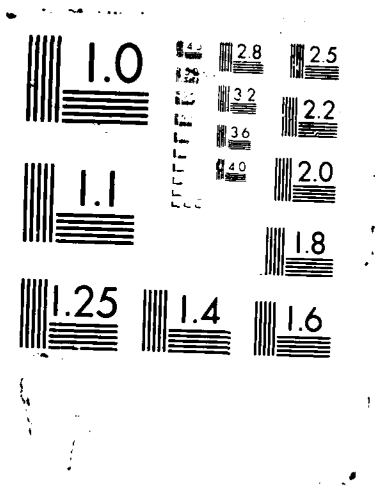
1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

DTIC FILE COPY

AD-A190 560

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AIM-966	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Fully Abstract Semantics for Event-based Simulation		5. TYPE OF REPORT & PERIOD COVERED AI-Memo
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Robert J. Hall		8. CONTRACT OR GRANT NUMBER(s) N00014-85-K-0124
9. PERFORMING ORGANIZATION NAME AND ADDRESS Artificial Intelligence Laboratory 545 Technology Square Cambridge, MA 02139		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE May, 1987
		13. NUMBER OF PAGES 17
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Office of Naval Research Information Systems Arlington, VA 22217		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

event-based simulation, denotational semantics, full abstraction

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Event-based simulation is a popular technique for predicting the behavior of, among other things, digital circuits. On the other hand, applicative denotational formalisms, in which circuits are represented by functional equations with an explicit time variable, are becoming popular for other reasoning tasks. Before a system is to use both approaches to modeling circuits, questions of semantic equivalence must be addressed. In particular, if two cir-

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

88 1 14 091

20. (cont)

cuits are equivalent in one formalism, will they be equivalent in the other? What modeling restrictions are needed to bring this about? This paper shows that, provided circuits contain no zero-delay loops, a tight relationship, full abstraction, exists between a natural event-based operational semantics for circuits and a natural denotational semantics for circuits based on causal functions on value timelines. The paper also discusses what goes wrong if zero-delay loops are allowed, and illustrates the application of this semantic relationship to modeling questions.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A.I. Memo No. 966

May, 1987

A Fully Abstract Semantics for Event-based Simulation

Robert J. Hall
M.I.T. Artificial Intelligence Laboratory
Cambridge, MA 02139

ABSTRACT. Event-based simulation is a popular technique for predicting the behavior of, among other things, digital circuits. On the other hand, applicative denotational formalisms, in which circuits are represented by functional equations with an explicit time variable, are becoming popular for other reasoning tasks. Before a system is to use both approaches to modeling circuits, questions of semantic equivalence must be addressed. In particular, if two circuits are equivalent in one formalism, will they be equivalent in the other? What modeling restrictions are needed to bring this about? This paper shows that, provided circuits contain no zero-delay loops, a tight relationship, *full abstraction*, exists between a natural event-based operational semantics for circuits and a natural denotational semantics for circuits based on causal functions on value timelines. The paper also discusses what goes wrong if zero-delay loops are allowed, and illustrates the application of this semantic relationship to modeling questions.

An abbreviated version of this paper will appear in the proceedings of the Fifteenth IASTED Conference on Applied Simulation and Modeling (ASM-87), Santa Barbara, CA.

Acknowledgments. This paper describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-85-K-0124.

1 Overview

Event-based simulation is a popular technique for predicting the behavior of digital circuits [8,10]. On the other hand, applicative denotational formalisms, in which circuits are represented by functional equations with an explicit time variable, are becoming popular for other reasoning tasks, *e.g.* hardware description [9], verification [4,1], and automatic generation of simulator models [5]. Before an integrated CAD system is to use both approaches to modeling circuits (for different purposes), the different representations must be shown to contain equivalent information. For example, it would be unsatisfactory if a circuit were "verified" using one representation, but also found to be *incorrect* by the simulator, using a different representation. Such a problem could arise from two sources: 1) the verifier or simulator could contain bugs, or 2) each is correct, but the underlying representations are fundamentally inequivalent. The purpose of this paper is to address the latter issue.

The approach used here is adapted from the literature on programming language theory, along the lines, for example, of [11]: define a mathematically precise meaning for the time function representation of a circuit and show that event-based simulation preserves this meaning. This approach has three steps.

First, a clean abstraction of event-based simulators is defined, giving circuits a clear operational semantics. It is hoped that the results proved with regard to this abstraction can be extended as necessary to cover a particular simulator.

Second, circuits are given denotational meanings by associating with each a particular mathematical function, mapping timelines (value histories) to timelines. This captures the semantics of applicative formalisms such as the "time functions" of [1].

Third, the two meanings, operational and denotational, are shown to be equivalent. That is, simulating the circuit for a given set of input events calculates the value of applying the function denoted by the circuit to the corresponding input timeline vector. This leads to the stronger result that the denotational semantics is *fully abstract* with respect to the operational semantics; that is, two circuits give the same simulation results on all inputs if and only if they denote the same timeline function.

This result has several applications. First, it shows that problems of the second type mentioned above can not occur in a system using representations based on the semantics given here. Second, it is useful in showing limitations of the the event-based formalism in modeling certain kinds of behaviors. Third, [5] describes a system which reasons denotationally to compose and simplify the functions of a circuit's components in order to produce a behavioral model for the circuit automatically. The equivalence of the representations is crucial to the soundness of the procedure.

In the discussion to follow, the term "circuit" is used to refer to a mathematical idealization of real circuits. It should be noted, however, that this idealization can be used to model many other real-time systems as well.

Gordon [4] defines a least-fixed-point denotational semantics for clocked, synchronous circuits. A major difference between that approach and this one is that this approach is able to model arbitrarily small time delays. Due to the synchrony of Gordon's model, he is able to define the meaning of a circuit as (essentially) a sequential (Mealy) machine. The current work is forced to more generality since input changes can happen arbitrarily close together in time. Gordon[4] does not address simulation issues.

Amblard, *et al* [1] give a formalism whose semantics is similar to the timeline semantics, but they do not attempt to relate it to any operational model. They give a scenario illustrating how human designers could use the formalism to verify circuits.

Meinen [9] gives an applicative formalism whose semantics is related to the present one. He makes reference to the automatic conversion of applicative descriptions to executable simulations, but does not address the connection between the two semantics.

2 Time, Values, and Circuits

The usefulness of the event-based simulation semantics presented here rests on three modelling assumptions: *the context independence assumption* requires that primitive components of the circuit have behaviors which are independent of the circuit in which they are used; *the digital approximation* assumes that it is possible consistently to map the observable values into a discrete set of values; and *the non-zero width event assumption* requires that changes in the discrete value of an observable persist for a positive duration.

Times and durations are modeled by the set $\mathbf{Q}^{+}$ of positive rationals.

That is, time starts at zero, but we may only observe the circuit values at positive rational times. The use of rationals instead of reals is a matter of convenience which simplifies some of the proofs.

Discrete event simulation maps observables (*e.g.* voltage) into a non-empty set $\mathbf{S}$ of *values* ("logic levels").

A circuit consists of a finite set of uniquely named nodes and a finite set of modules connected to the nodes through named ports. The node (an abstraction of "wire," "bus," "net," etc.) is intended to represent a place which holds a value during the course of computation. A module represents a computing element with zero or more input ports and exactly one output port. As such it has a type which defines the input/output relationship ("function"). The connections associate nodes with ports of modules. No more than one output port may be connected to any single node. Nodes must have some port connected to them. Nodes with no output port connected will be referred to as circuit inputs, as they must be driven from the outside at all times. A node is termed an input to a module if some input port of the module is connected to the node. Similarly, a node is an output of a module if the output port of the module is connected to the node.

Allowed modules are of two basic types. First, fix a convenient subset. $\mathcal{F} \subseteq \cup_{k\geq 0}\{f \mid f : \mathbf{S}^k \rightarrow \mathbf{S}\}$. (Zero-ary functions are constants.) The first type of primitive module is a pair $(f, \epsilon) \in \mathcal{F} \times \mathbf{Q}^+$. Intuitively, this represents the module which puts out $f(\vec{x})$ to its output port at time $t + \epsilon$ whenever its input ports have values $\vec{x}$ at time t. f_ϵ denotes a primitive module of this type, where f is a function identifier and ϵ is a duration (delay). The second primitive type of module is the "perfect memory element." Such a module type is denoted $M(\chi, \epsilon)$, where $\chi \subseteq \mathbf{S}$ and $\epsilon \in \mathbf{Q}^+$. It has two input ports, s and a. Its output port is defined to produce at time $t + \epsilon$ the value of a at the most recent time $u \leq t$ such that $s \in \chi$ at time u. For example, $M(\{1\}, 1.0)$ is an idealization of a D-flipflop with 1.0 unit of delay.

3 Operational Semantics

This section defines an event-based operational semantics for circuits. In particular, an effective evaluator, EVAL, is given which calculates the value of a given circuit node at a given simulated time in response to a given input specification.

A *simulation program*, p, is a quintuple $(w_p, J_p, I_p, y_p, t_p)$, where w_p is the circuit to be simulated; J_p is an initialization mapping which gives initial (time 0) values for all of the non-input nodes of w_p; I_p is a finite collection of all events which will ever occur on the input nodes of w_p (it must contain exactly one time 0 event for each input); y_p is the non-input node of w_p whose simulated value we wish to measure at simulated time t_p.

This semantics is based on the idea of *events* occurring at given nodes at given times. An event is a triple (y, t, v), which represents the change of node y's value to v at time t. The value of a node at any (simulated) time is simply the value of the most recent event for that node to occur before that time.

When an event occurs at time t on an input node of a module of type f_ϵ with input nodes $\vec{x}$, a new event is scheduled for the output node at time $t + \epsilon$ with value $f(\vec{x})$, where $\vec{x}$ refers to the new values of the inputs at time t. When an event occurs at either of the input nodes of a module of type $M(\chi, \epsilon)$, then if the s input's (new) value lies in χ at t an event is scheduled at time $t + \epsilon$ to set the output node to the (new) value of the a input at t.

EVAL is defined as follows. A simulation program, p, defines an abstract machine, $\rightarrow_p$, which operates on pairs (e, s) of (event-set, store), called machine-states. Intuitively, e represents the set of those events which are scheduled to happen on circuit nodes, but which have not yet occurred. s maps each node to the value of the most recent event to have occurred for that node. For any machine-state m, define $\text{cet}(m)$, the current-event-time of m, as the minimum time of any event in e_m. (If e_m is empty, then $\text{cet}(m) = \infty$.) Denote by $ims(p)$ (initial machine-state) the pair (e_0, s_0), where s_0 maps each non-input node, x, into $J_p(x)$ and each input node into the value of the time 0 event for that node in I_p. e_0 is I_p augmented with events setting each non-input node to its J_p value at time 0.

Define $prop_{w_p}(N, s, t)$, for N a set of nodes of w_p, s a store for w_p, and $t \in \mathbf{Q}^+$, to be the set of events constructed as follows. For each module of type f_ϵ with input nodes $\{x_i\}$ and output node y such that for some i, $x_i \in N$, form the event $(y, t + \epsilon, f(s(x_1), \ldots, s(x_k)))$. For each module of type $M(\chi, \delta)$ with inputs x_s and x_a and output y such that at least one of $x_s, x_a \in N$, if $s(x_s) \in \chi$, then form the event $(y, t + \delta, s(x_a))$.

Define $(e, s) \rightarrow_p (e', s')$ as follows. Let e_0 be the set of events in e with time $\text{cet}(e, s)$. s' is s updated with new values for nodes having an event in e_0. Let CN be the set of nodes, x, such that $s'(x) \neq s(x)$. $e' = (e - e_0) \cup$

$prop_{\omega_p}(\mathrm{CN}, s', \mathrm{cet}(\epsilon, s))$.

$\rightarrow_p$ is applied iteratively, starting with $ims(p)$, until the first machine-state, x, is reached with $\mathrm{cet}(x) \geq t_p$. At that point, evaluation terminates with $\mathrm{EVAL}(p) = s_x(y_p)$.

Theorem. EVAL is well-defined for all simulation programs.
Proof. One shows by induction on $\rightarrow$ steps that, *once all of the (finitely many) input events are specified*, the current-event-time of the intermediate machine-states strictly increases and is always an integer multiple of $\delta > 0$, where δ is one over the least common denominator of the input event times and the delays in the circuit. t_p must be reached after no more than $\lceil t_p/\delta \rceil$ steps. Note that δ depends on the input events, so the circuit alone is not just equivalent to a Mealy machine with clock period δ. □

4 Denotational Semantics: Σ

This section defines a semantical structure, Σ, together with a meaning mapping, $[[\cdot]]$, which associates denotations with circuits.

A *half-timeline on* $\mathbf{S}$ *is defined to be a map* $\rho : \mathbf{Q}^+ \rightarrow \mathbf{S}$ *which is piecewise* constant and obeys the *right-hand endpoint convention*. That is, for every point $t \in \mathbf{Q}^+$, there is a $\delta > 0$ such that ρ is constant on $[t - \delta, t]$. We also assume that for every $a > 0$, there are a finite number of transition points of ρ in the interval $(0, a)$. Denote the set of all half-timelines on $\mathbf{S}$ by $\mathbf{H}^1(\mathbf{S})$. (When the choice of $\mathbf{S}$ is clear from context, the explicit reference to $\mathbf{S}$ may be omitted.) It should be clear that, for any $k > 0$, there is a natural isomorphism between $(\mathbf{H}^1(\mathbf{S}))^k$, the k-fold cross-product of $\mathbf{H}^1(\mathbf{S})$ with itself, and the set $\mathbf{H}^1(\mathbf{S}^k)$. We shall therefore make no distinction between the two. For any $k > 0$, let $\mathbf{H}^k(\mathbf{S})$ denote the set $\mathbf{H}^1(\mathbf{S}^k)$.

The use of value timelines (in various slightly different forms) to model real-time behavior is ubiquitous in the literature on real-time programs and not uncommon in the literature on circuits [2,3,7]. Most use a discrete time domain, rather than the rationals or reals. (Meinen [9] uses the reals, but also assumes there is a global "minimum cycle time," making the model equivalent to a discrete time domain.)

Definition. For $m, n > 0$, a function $f : \mathbf{H}^m \rightarrow \mathbf{H}^n$ is said to be *causal* if and only if there exists a positive rational ϵ_f such that the following holds.

For $\rho, \rho' \in \mathbf{H}^m$ and for all $t > 0$,

$$\exists v \in (0, t + \epsilon_f].f\rho(v) \neq f\rho'(v) \Longrightarrow \exists u \in (0, t].\rho(u) \neq \rho'(u).$$

For $m, n > 0$, let $\mathbf{CF}^{m \to n}$ denote the set of all causal functions $f : \mathbf{H}^m \to \mathbf{H}^n$. $\mathbf{CF}^{0 \to n}$ is identified with $\mathbf{H}^n$.

Definition. $\Sigma =_{\text{def}} (\mathbf{Q}^+, \mathbf{S}, \cup_{k>0}\mathbf{H}^k, \cup_{m \geq 0, n>0}\mathbf{CF}^{m \to n})$.

Let $\pi_i : \mathbf{S}^k \to \mathbf{S}$ denote the projection onto the ith coordinate. This induces a (non-causal) function $\pi_i : \mathbf{H}^k \to \mathbf{H}^1$ pointwise. Suppose $\rho \in \mathbf{H}^m, \rho' \in \mathbf{H}^n$. Let $[\rho, \rho']$ denote that element of $\mathbf{H}^{m+n}$ whose value at any t has first m components equal to those of $\rho(t)$ and last n components equal to those of $\rho'(t)$. Similarly, if $f \in \mathbf{CF}^{k \to m}, g \in \mathbf{CF}^{k \to n}$, let $[f, g]\rho =_{\text{def}} [f\rho, g\rho]$. The following facts are immediate.

Lemma.

- If f is causal, then $\pi_i f$ is causal for any i.
- If f, g are both causal, then $f[g, \text{id}]$ is causal, where id is the identity function on the appropriate $\mathbf{H}^i$ to make the argument to f be of the correct type. Similarly, $f[\text{id}, g]$ is causal.
- If $f \in \mathbf{CF}^{k \to m}, g \in \mathbf{CF}^{k \to n}$ are causal, then $[f, g]$ is causal.
- Let $f \in \mathbf{CF}^{m \to n}$. Then for any $k > 0$, there exists a unique extension $\tilde{f} \in \mathbf{CF}^{m+k \to n}$, such that for all $\rho \in \mathbf{H}^m, \rho' \in \mathbf{H}^k, \tilde{f}[\rho, \rho'] = f\rho$.

□

Definition. Let $t \in \mathbf{Q}^+$. Define $\mathbf{H}^k{}_t$, the set of all *t-initial half-timelines on* $\mathbf{S}^k$, as

$$\{\rho_t : (0, t] \to \mathbf{S}^k \mid \exists(\rho \in \mathbf{H}^k).\forall(0 < u \leq t).\rho(u) = \rho_t(u)\}$$

If $f \in \mathbf{CF}^{m \to n}$, then let f_t denote the pointwise-induced function $f_t : \mathbf{H}^m{}_t \to \mathbf{H}^n{}_t$. That is, $f_t\rho_t(s) = f\rho(s)$ for all $0 < s \leq t$.

Lemma. Let $f \in \mathbf{CF}^{m \to n}$. Then for all $t \in \mathbf{Q}^+$, f induces a function $\hat{f}_t : \mathbf{H}^m{}_t \to \mathbf{H}^n{}_{t+\epsilon_f}$, by setting

$$\hat{f}_t \rho_t = (f\rho)_{t+\epsilon_f},$$

where ρ is any extension of ρ_t.

Proof. Let ρ, ρ' be any extensions of ρ_t. Since f is causal, $f\rho$ agrees with $f\rho'$ for all times less than or equal to $t + \epsilon_f$. □

Theorem (Fixed Point Theorem). Let $f \in \mathbf{CF}^{n \to n}$. There exists a unique $\rho \in \mathbf{H}^n$, denoted μf, such that $\rho = f\rho$.

Proof: The plan is to show that for any $t \in \mathbf{Q}^+$, there is a unique t-initial half-timeline, ρ_t, which satisfies $\rho_t = f_t \rho_t$, and that all such agree on their areas of overlap. That done, we simply define ρ to be the unique half-timeline which agrees with all of them. That is, for any t, let $\rho(t)$ be the common value at t of all the $(t + \delta)$-initial half-timelines, for $\delta \geq 0$. This is certainly unique; that it satisfies the axioms for being a half-timeline is easily seen to be inherited from the t-initial ones from which it is constructed.

First, if there exists a *unique* such t-initial half-timeline for every t, then they must all agree on their areas of overlap. Suppose ρ_t satisfies the equation, and ρ'_s also satisfies, and $s < t$. But then ρ_s must also satisfy, since ρ_t satisfies pointwise for all times less than or equal to t. By the uniqueness property, we see $\rho'_s = \rho_s$. Thus, the original two agree on their area of overlap.

Now, suppose $t \leq \epsilon_f$. Then since f is causal, f_t must be a constant function. Clearly, $(f_t \nu_t) = f_t(f_t \nu_t)$ for any ν, and $f_t \nu_t$ is the unique such solution.

Now, let $t = k\epsilon_f$, $k \geq 1$ an integer, and suppose it is the case that for every $u \leq t$ there exists a unique ρ_u satisfying the equation at time u. Let $v \in (t, t + \epsilon_f]$. Define $\rho_v = \hat{f}_{v-\epsilon_f} \rho_{v-\epsilon_f}$.

Clearly, $\rho_v = (f\rho)_v = f_v \rho_v$, from the definitions. Is ρ_v the unique solution? For any solution ρ'_v of the equation, we have

$$\rho'_v = f_v \rho'_v = (f\rho')_v = \hat{f}_{v-\epsilon_f} \rho'_{v-\epsilon_f},$$

where $\rho'_{v-\epsilon_f}$ must also satisfy the equation at time $v - \epsilon_f$. But by the induction hypothesis, $\rho'_{v-\epsilon_f} = \rho_{v-\epsilon_f}$. Plugging that into the above, we see that $\rho'_v = \rho_v$.

Since v was arbitrary in the half-open interval $(k\epsilon_f, (k+1)\epsilon_f]$, we have verified the induction hypothesis for all times less than or equal to $(k+1)\epsilon_f$, and the induction is complete. □

Before assigning meanings, it is useful to show that causal functions are closed under finite *wiring diagram composition*. This is the sort of composition one finds in a circuit: normal functional composition, together with fixed point operations as required by feedback loops.

Definition. Given $\{f_i \in \mathbf{CF}^{m+n \to 1} \mid 1 \leq i \leq m\}$, let $F = [f_1, \ldots, f_m]$, the "vectorization" of the f_i. (F is causal by a previous Lemma.) Define $\bigodot_i f_i : \mathbf{H}^n \to \mathbf{H}^m$, the *wiring diagram composition* of the $\{f_i\}$, by $(\bigodot_i f_i)(\nu) = \mu(F[\mathrm{Id}, \nu])$.

Lemma. Let $F \in \mathbf{CF}^{m+n \to m}$; $\nu, \nu' \in \mathbf{H}^n$ such that $\nu_t = \nu'_t$ for some $t \in \mathbf{Q}^+$. Let $\rho = (\bigodot_i(\pi_i F))\nu$ and $\rho' = (\bigodot_i(\pi_i F))\nu'$. Then $\rho_t = \rho'_t$.
Proof: $\rho_t = (F[\mathrm{Id}, \nu])_t \rho_t = F_t[\rho_t, \nu_t] = F_t[\rho_t, \nu'_t] = (F[\mathrm{Id}, \nu'])_t \rho_t$. However, ρ'_t is the unique solution to the fixed-point equation for $(F[\mathrm{Id}, \nu'])_t$ (see proof of Fixed Point Theorem). Thus, $\rho_t = \rho'_t$. □

Theorem (Wiring Diagram Composition). Let $f_i \in \mathbf{CF}^{m+n \to 1}$, for $i = 1 \ldots m$. Then $\bigodot_i f_i \in \mathbf{CF}^{n \to m}$.
Proof: Let $t \in \mathbf{Q}^+$, and suppose $\nu, \nu' \in \mathbf{H}^n$ agree on their t-initial segments. Let $\rho = (\bigodot_i f_i)\nu$, $\rho' = (\bigodot_i f_i)\nu'$, and $F = [f_1, \ldots, f_m]$. Then

$$\begin{aligned}
\rho_{t+\epsilon_F} &= (F[\rho, \nu])_{t+\epsilon_F} && ;\rho \text{ is a fixed point} \\
&= \hat{F}_t[\rho, \nu]_t && ;\text{def of } \hat{F}_t \\
&= \hat{F}_t[\rho', \nu']_t && ;\text{previous Lemma and hypothesis} \\
&= (F[\rho', \nu'])_{t+\epsilon_F} && ;\text{def of } \hat{F}_t \\
&= \rho'_{t+\epsilon_F} && ;\rho' \text{ is a fixed point}
\end{aligned}$$

□

Meanings are assigned in Σ to initialized circuits, (w, J), as follows. Suppose w has m non-input nodes and k input nodes. Order the nodes via the function o : nodes $\to \{1 \ldots m+k\}$, letting the non-inputs come before the inputs. Each primitive module in (w, J) has a natural interpretation as a causal function. If it is of type f_ϵ and its output node is y, then let $c_{o(y)}$, the causal function for the module driving node y, be given by $(c_{o(y)}\vec{\rho})(t) = f(\vec{\rho}(t-\epsilon))$ if $t > \epsilon$, $J(y)$ otherwise. If the module driving node y is of type $M(\chi, \epsilon)$, then $(c_{o(y)}[\rho_a, \rho_s])(t) = \rho_a(u)$, where u, if it exists, is the most recent time $\leq t - \epsilon$ such that $\rho_s(u) \in \chi$; if u does not exist, then $c_{o(y)}$ has value $J(y)$.

Definition. $[[(w, J)]] : \mathbf{H}^k \rightarrow \mathbf{H}^m$ is defined to be $\bigodot_i c_i$.

Corollary. $[[(w, J)]] \in \mathbf{CF}^{k \rightarrow m}$. $\square$

Definition. For a simulation program, p, define

$$[[p]] = (\pi_{o(y_p)}[[(w_p, J_p)]])([[I_p]])(t_p),$$

where $[[I_p]]$ is the obvious timeline vector constructed from the event-set I_p.

5 Connections

Theorem (Computational Adequacy). For any simulation program p, $[[p]] = \text{EVAL}(p)$.
Proof. It suffices to show that there is some ρ_{t_p} such that $\pi_{o(y_p)}\rho_{t_p}(t_p) = \text{EVAL}(p)$ and $\rho_{t_p} = C_{t_p}[\rho_{t_p}, [[I_p]]_{t_p}]$, where $C = [c_1, \ldots, c_m]$.

Suppose $ims(p) \rightarrow_p^* x$. Define $h_p(x)$, the *event-history for p at x*, inductively as follows. $h_p(ims(p)) = \text{cet-events}(e_{ims(p)})$, where cet-events($e$) is defined to be the set of events of e with time cet(e). If $x' \rightarrow_p x$ then $h_p(x) = h_p(x') \cup \text{cet-events}(e_x)$.

For any simulation program p define the function $\text{val}_p : \text{nodes} \times (0, t_p] \rightarrow \mathbf{S}$ as follows. Let x be the machine-state whose cet is minimum, but greater than or equal to t_p, such that $ims(p) \rightarrow_p^* x$. Then $\text{val}_p(y, t)$ is the value part of the latest event for y in $h_p(x)$ whose time is strictly less than t. (It is easy to see via induction on $\rightarrow_p$ steps that $h_p(x)$ is simply a record of all events which have occurred with times less than or equal to t_p.) It follows from the definition of EVAL that $\text{val}_p(y, t_p) = s_x(y) = \text{EVAL}(p)$. Moreover, val_p defines a t_p-initial half-timeline by vectorizing the individual functions $\text{val}_p(y, \cdot)$ in the order o.

Thus, it suffices to show for each non-input node y (with associated module m whose inputs are the nodes x_i)

$$\text{val}_p(y, \cdot) = c_y(\text{val}_p(x_1, \cdot), \ldots).$$

The only way c_y could fail to be satisfied at some time is if an event occured that changed an input or output node to an incompatible value. But whenever an input event occurs, $prop_{w_p}$ dictates that an output event be scheduled at exactly the correct delay to maintain consistency of the constraint. Conversely, output events are scheduled only as results of input events; hence, the output event must maintain consistency with c_y. □

This theorem shows that the value denoted by a simulation program is always the same as that computed by the event-based simulation. This allows us to reason about a simulation using the denotations of the circuits, instead of thinking about how the simulator will propagate events. This is most useful as a means to proving the stronger result (below) that a pair of *initialized circuits* (as distinct from their uses in simulation programs) are behaviorally equivalent if and only if they are denotationally equivalent.

Theorem (Full Abstraction). Suppose that (w, J) and (w', J') have the same set of inputs. Then

$$\forall I \forall t . \mathrm{EVAL}(w, J, I, y, t) = \mathrm{EVAL}(w', J', I, y', t)$$

if and only if

$$\pi_{o(y)}[[(w, J)]] = \pi_{o'(y')}[[(w', J')]].$$

Proof. *(Only if)* The values of causal functions on inputs with infinitely many value changes must be compatible with their values on inputs having finitely many value changes. (This is because there exists an eventually constant completion of any t-initial segment of a half-timeline. The t-initial segment of the value of the causal function is not affected by the difference in the tail of the input timeline.) The hypothesis and computational adequacy imply that the two functions are equal on all inputs with only finitely many value changes.

(If) This direction follows immediately from computational adequacy and the definition of meaning of simulation programs. □

Theorem (Modularity). Let $c_1, \ldots, c_{k+m+n} \in \mathbf{CF}^{k+m+n+l \to 1}$, and let $s \in \mathbf{CF}^{n+l \to k+m}$ be defined by $s = \bigodot_{i=1 \ldots k+m} c_i$. Denote by $w \in \mathbf{CF}^{l \to k+m+n}$ the function $\bigodot_{i=1 \ldots k+m+n} c_i$. Suppose that $c_{k+m+1}, \ldots, c_{k+m+n}$ do not depend on the first k components of the timeline vector. Then

$$(\pi_{k+1} \theta s) \odot \ldots \odot (\pi_{k+m} \theta s) \odot \eta c_{k+m+1} \odot \ldots \odot \eta c_{k+m+n} = [\pi_{k+1} w, \ldots, \pi_{k+m+n} w],$$

where $\theta s \in \mathbf{CF}^{m+n+l \to k+m}$ is defined by $\theta s(\rho) = s[\pi_{m+1}\rho, \ldots, \pi_{m+n+l}\rho]$, and $\eta c_{k+m+i} \in \mathbf{CF}^{m+n+l \to 1}$ is defined by $\eta c_{k+m+i}(\rho) = c_{k+m+i}[*, \ldots, *, \pi_1\rho, \ldots, \pi_{m+n+l}\rho]$. ($*$ denotes some (any) particular timeline.)
Proof. This follows immediately from the uniqueness of fixed-points. $\square$

This just says that the function of a subcircuit is preserved in the context of a larger circuit, and that "internal" nodes of the subcircuit (nodes $1 \ldots k$ above) are important only to the subfunction.

Corollary. Replacement of a subcircuit by a denotationally equivalent implementation has no effect on overall circuit behavior. $\square$

6 Extension: Zero-delay

It is not difficult to extend these results to cover circuits containing primitive modules with zero delay from input to output, *assuming the circuits contain no zero-delay feedback loops.* The functions obtained are no longer necessarily causal functions, of course, but one can still show that if the simulator orders the processing of events properly, the evaluation will yield the unique, correct solution to the recursion equations. This larger class of functions, however, is not closed under arbitrary composition, because it is possible to connect legal zero-delay subcircuits into a zero-delay feedback loop.

The problem with allowing zero-delay loops in circuits is illustrated by an example: the circuit with two zero-delay inverters connected in a ring, initialized with its two nodes at opposite logical values. Since zero-delay circuits are idealizations of circuits with positive delay (which have non-trivial behaviors), one presumably wishes the simulator to stop propagating events once all the node values are consistent, allowing simulation to proceed.

In terms of denotational semantics, however, the fixed-point equation corresponding to this circuit has infinitely many solutions. (All timelines satisfy $y(t) = \neg\neg y(t)$ for all t.) But then the semantics is no longer fully abstract: *the simulator fails to compute all of the possible behaviors of the circuit.*

(Gordon, 1981) defines a denotational semantics for circuits which models zero-delay loops. Any zero-delay loop denotes $\perp$, the symbol for divergence. This is correct in the case of, for example, a 1-inverter ring, as one would expect the simulation to fail to halt in that case. Unfortunately, the 2-inverter ring also denotes $\perp$, and hence Gordon's (1981) denotational semantics fails to be computationally adequate for a simulator which converges on the 2-inverter circuit.

The lack of full abstraction due to zero-delay loops can cause problems to systems which use both a simulator representation and a denotational representation. Consider a system which manipulates circuit designs by replacing subcircuits with different implementations to achieve some performance improvement. Suppose that it is allowed to replace one subcircuit by another if the two denote precisely the same set of timeline functions. This would allow it to replace a two-inverter loop, with nodes a and b initialized to $\{(a,0),(b,1)\}$, by a two-inverter loop initialized to $\{(a,1),(b,0)\}$. These circuits have precisely the same set of fixed points, namely $\mathbf{H}^1$. On the other hand, our simulator evaluates these quite differently: the first produces a constant 1 for all times, the second produces a constant 0. Thus, the system could make a transformation which failed to preserve operational behavior.

It may be possible to find a denotational semantics for circuits which is fully abstract for the zero-delay loop simulator, but that is beyond the scope of this paper.

7 Applications

The applicative denotational formalisms, for which Σ provides a precise mathematical meaning, seem to be well-suited to various forms of reasoning about circuits, both by humans and machines [1,4,9]. In particular, they are highly local and make explicit the relevant time dependencies between values [5], properties crucial to reasoning about the function of circuits. On the other hand, event-based simulation is well established as a useful technique for predicting the behavior of circuits. The results in this paper provide a formal proof that designers can employ tools which use these different representations and still obtain coherent results. This is the chief contribution of this paper.

Another important contribution, however, is that the denotational semantics can be used to better understand the computational technique of event-based simulation. For example, it is well-known that modeling certain low-level circuit devices, like MOS transistors and bi-directional buses, is difficult and seems to require additional simulator formalism (such as extra

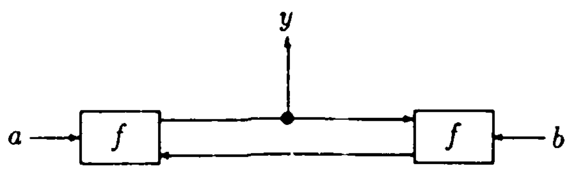

Figure 1: A pair of buses connected together, hopefully to form a larger bus.

port types in addition to just "input" and "output"). Using the denotational semantics, one can *prove* that certain devices can not be modeled in the event-based formalism given here.

To illustrate the technique, consider a modeling scheme in which node values have a strength aspect, "driven" or "undriven," in addition to a logical value. (This is sometimes done to handle stored charge.) "Driven" means roughly that the node is connected to a power source, while "undriven" means the node is merely storing charge. More precisely, suppose there exists a function, $p : \mathbf{S} \rightarrow \{0,1\}$, such that $ps = 1$ if and only if s stands for a driven value. Note that $\mathbf{S}$ may have more than two elements; even infinitely many.

The question is, does their exist some $\mathbf{S}$ and some circuit, expressed as a combination of the given primitives, that models the behavior of a bidirectional bus? We will show that the answer is no by stating the axioms we wish the bus to obey and then showing them to be inconsistent with the structure equation for the circuit shown in Figure 1. We choose the following bus axioms.

- The bus should have two inputs, a and b, and one output, y. y should be driven at time t if input a is driven at time $t-\epsilon_1$ or if b is driven at $t-\epsilon_2$, or both. More precisely, if $y = f(a,b)$, then $pf(a,b) = \mathrm{OR}_{\epsilon_1,\epsilon_2}(pa, pb)$, where $\epsilon_1, \epsilon_2 > 0$.

- When two buses are connected together, as shown in Figure 1, the conglomerate should act as one bus, possibly with different delays. That is, denoting the overall function (as seen at output y in the figure) by f', we have $pf'(a,b) = \mathrm{OR}_{\epsilon_3,\epsilon_4}(pa, pb)$.

- From the structure of the circuit, we can also derive the equation

$$f'(a,b) = f(a, f(b, f'(a,b))).$$

It will suffice for this argument to assume that $pb(t) = 0$ for all t. Using this assumption to substitute and simplify the equations above, it is not difficult to derive the equation

$$z_{\epsilon_3}(pa) = \mathrm{OR}_{\epsilon_1, 2\epsilon_2+\epsilon_3}(pa, pa),$$

where $z_\epsilon(y)(t)$ is defined to be $y(t-\epsilon)$. But this clearly does not hold for all choices of a: let a be driven for some interval and then go to undriven thereafter. Thus, the bus axioms are inconsistent with the structure axioms, and so no such f and f' can exist which model the bus in this way.

8 Summary and Conclusions

This paper has defined an event-based operational semantics for circuits and a fully abstract denotational semantics, Σ, based on causal functions on timelines. The principle results are

- Causal functions on half-timelines satisfy the following.
  - For all $n > 0$, every function in $\mathbf{CF}^{n \to n}$ has a unique fixed point.
  - Causal functions are closed under arbitrary finite composition (*i.e.* arbitrary wiring diagrams).
- EVAL always terminates.
- Σ is computationally adequate and fully abstract for simulation in that two initialized circuits behave the same if and only if they denote the same function.
- A circuit behaves the same way when embedded in a larger circuit as it does in isolation.
- *Extension to Zero-delay elements:* Computational adequacy is lost if zero-delay loops are allowed, but a kind of extension is possible if such loops are disallowed.

The previous section discussed the principle uses of these results: they give a formal justification for using different representations of circuits in the same CAD system, and they provide insight into the limitations and applications of the event-based computational technique. Future research questions include

- How can this semantics be extended to capture bi-directional busses, pass transistors and other low level elements?
- Currently, the formalism allows only primitive functions with fixed delays from inputs to output. This is no loss of generality if **S** is finite, but if we allow **S** to be infinite, can we extend the results to primitives with variable delay? Can we then extend it to capture the semantics of real-time computer networks [6,3]?

Acknowledgments

The author wishes to thank Albert Meyer for his helpful criticism of earlier drafts of this paper. This paper describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-85-K-0124.

References

[1] Amblard, P., Caspi, P., and Halbwachs, N. (1985) Describing and reasoning about circuits behaviour by means of time functions. In *Proceedings of the 7th International IFIP Conference on Computer Hardware Description Languages and their Applications*. Koomen and Moto-oka (eds.) Elsevier Science Publishers B.V. (North-Holland).

[2] Arvind, and Brock, J. D. (1984). Resource managers in functional programming. *Journal of Parallel and Distributed Computing* **1**, 1984.

[3] de Bruijn, A., and Bohm, W. (1985) The denotational semantics of dynamic networks of processes. *ACM Transactions on Programming Languages and Systems*. **7** (4) October, 1985.

[4] Gordon, M.J.C. (1981) Register transfer systems and their behaviour. In *Proceedings of the Fifth International IFIP Symposium on Computer Hardware Description Languages and their Applications*. Breuer and Hartenstein (eds.) North-Holland Publishing Co.

[5] Hall, Robert J., Lathrop, Richard H., and Kirk, Robert S. (1987) A multiple representation approach to understanding the time behavior of digital circuits. To appear in *Proceedings of the Sixth National Conference on Artificial Intelligence, AAAI-87*, Seattle, Washington.

[6] Kahn, G. (1974). The semantics of a simple language for parallel programming. In *Proceedings IFIP 74*. J.L. Rosenfeld (ed.). North-Holland, Amsterdam, 1974.

[7] Kelly, V.E. (1984) The CRITTER system – automated critiquing of digital circuit designs. In *Proceedings of the 21st IEEE Design Automation Conference*, Albuquerque, NM.

[8] Lathrop, Richard H., and Kirk, Robert S. (1985) An extensible object-oriented mixed-mode functional simulation system. In *Proceedings of the 22nd IEEE Design Automation Conference*. Las Vegas, NV.

[9] Meinen, P. (1979) Formal semantic description of register transfer language elements and mechanized simulator construction. In *Proceedings of the 4th International Symposium on Computer Hardware Description Languages*. Palo Alto, CA.

[10] Mentor Graphics Corporation. (1986). *Behavioral Language Model™ (BLM) User's Manual*. Beaverton, OR: Mentor Graphics Corporation.

[11] Plotkin, G.D. (1977) LCF considered as a programming language. *Theoretical Computer Science* **5**, pp. 223-255. North-Holland.